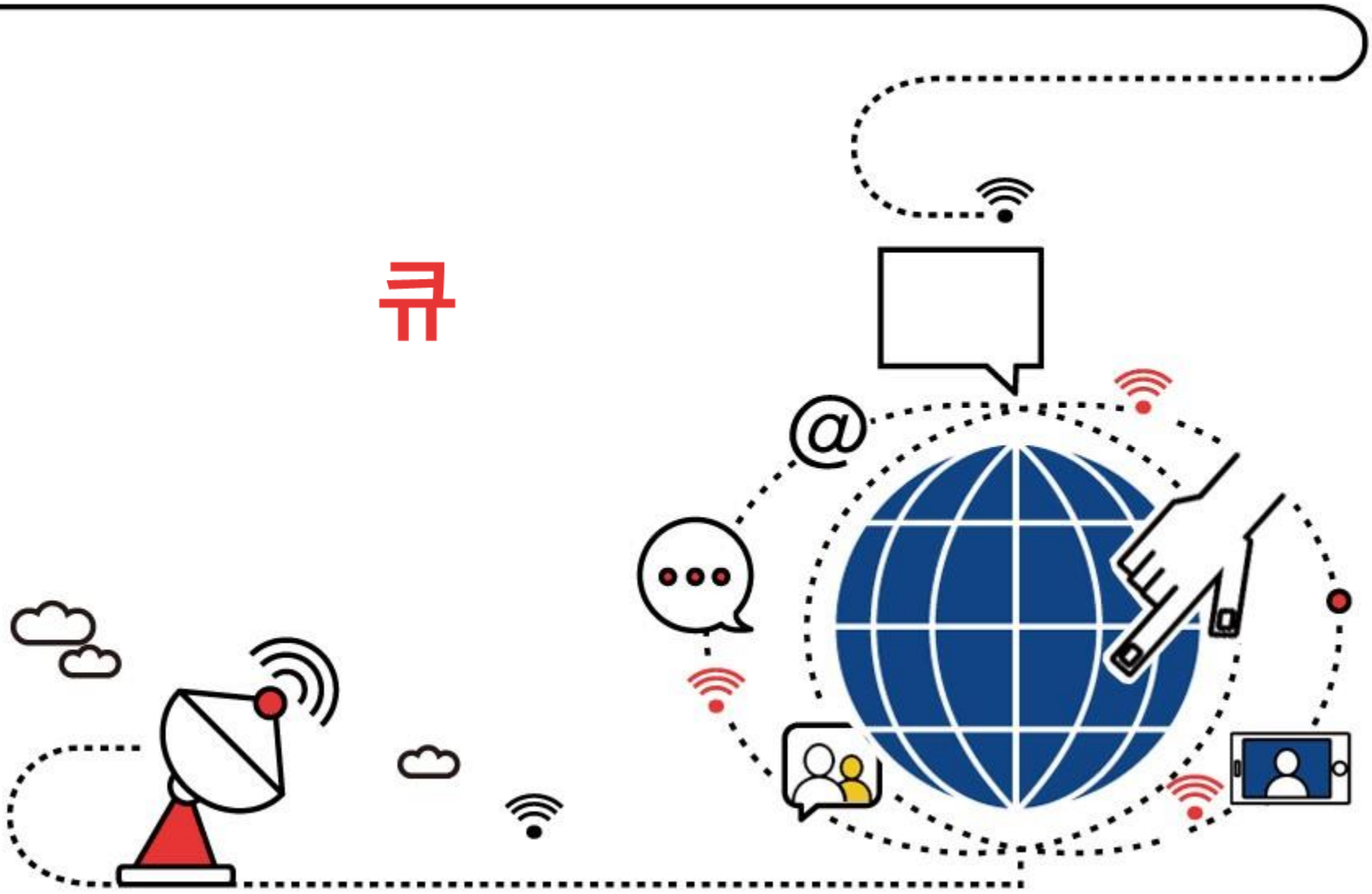
큐



1 큐

1 First-In First-Out : FIFO

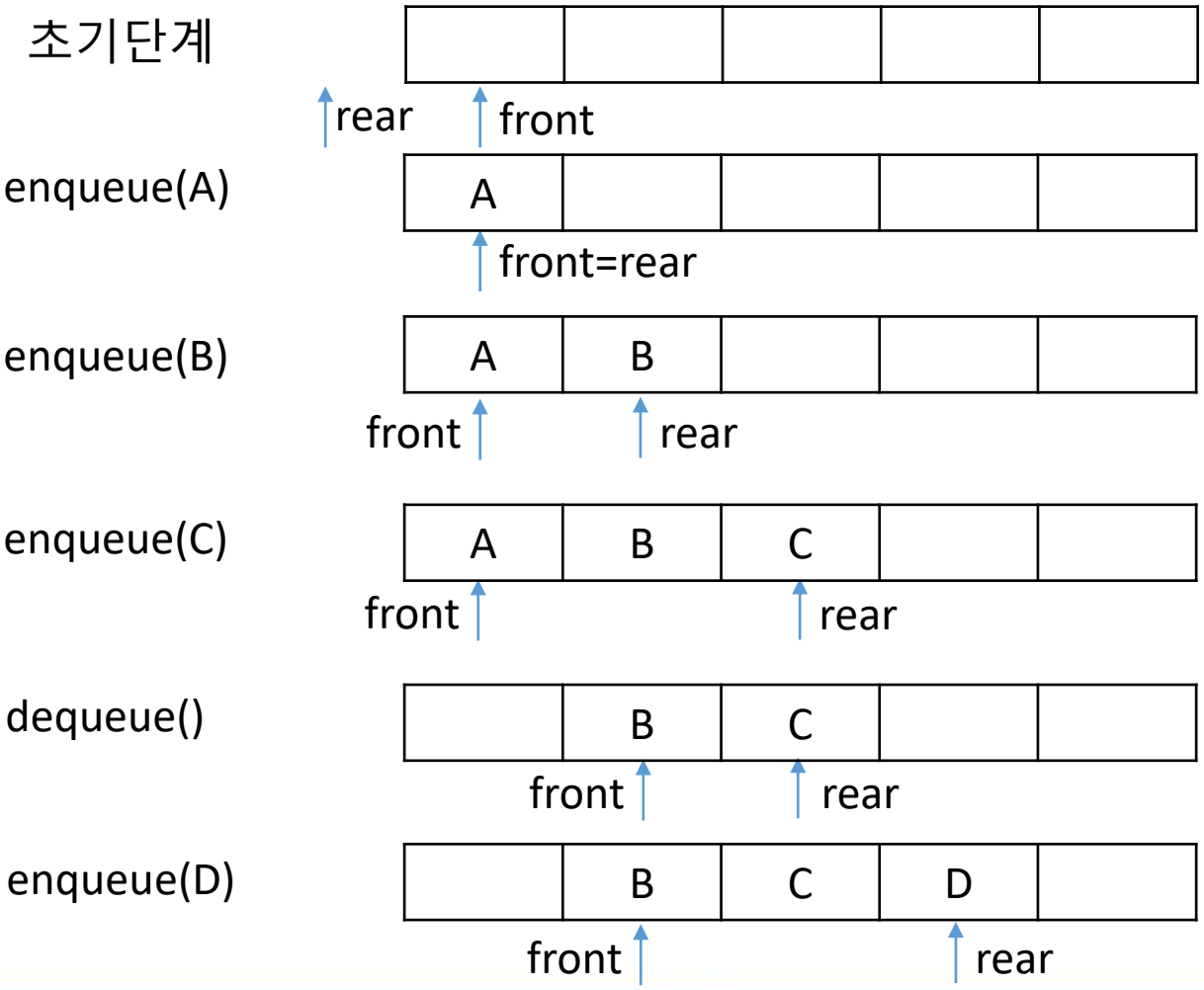
- 처음 들어간 것이 먼저 나오도록 되어있는 자료 구조.
  원소가 rear을 통해서 삽입되고, front를 통해 삭제됨.

2 추상 데이터 타입

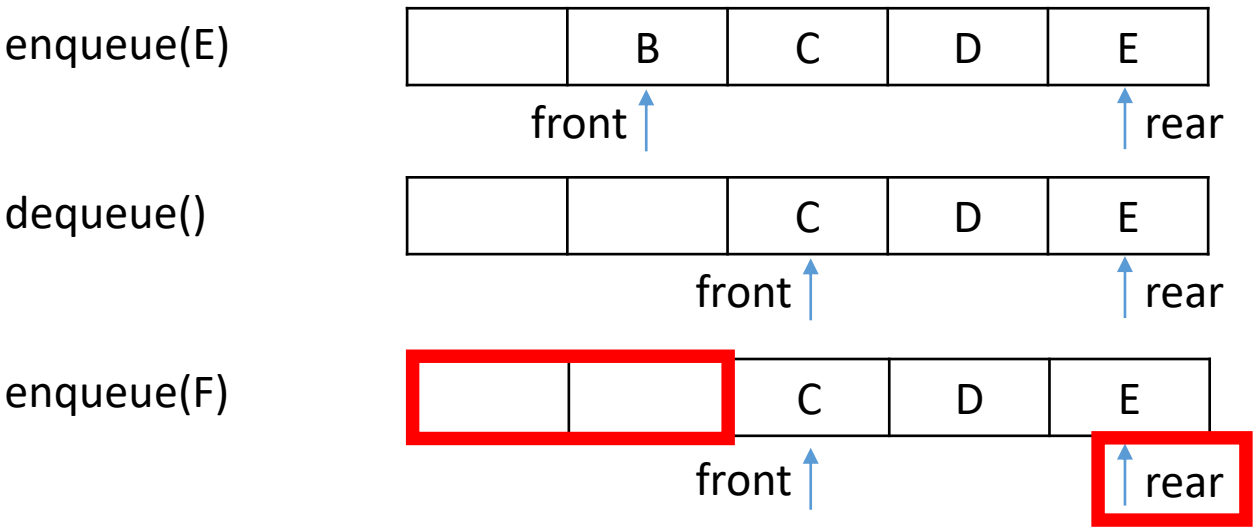
- front : 원소의 접근 및 삭제는 처음 원소에서 일어나는 것으로 제한됨.
- rear : 원소의 삽입은 큐의 가장 뒤에서 일어나는 것으로 제한됨.

- enqueue(e) : 큐의 rear에 객체 e를 삽입
- dequeue() : 큐의 front에서 객체를 삭제. 빈 큐일 경우 에러를 발생시킴
- front() : 큐의 맨 앞의 값(가장 먼저 입력한 것)을 제거하지 않고 참조만을 반환.
- rear() : 큐의 맨 뒤의 값을 제거하지 않고 참조만을 반환.
  >> front(), rear()는 큐가 비었을 경우 에러를 발생시킴.
- size() : 큐 안의 객체의 개수를 반환
- isEmpty() : 큐가 비었으면 참, 그렇지 않으면 거짓을 반환

3 배열기반 선형 큐의 삽입과 삭제

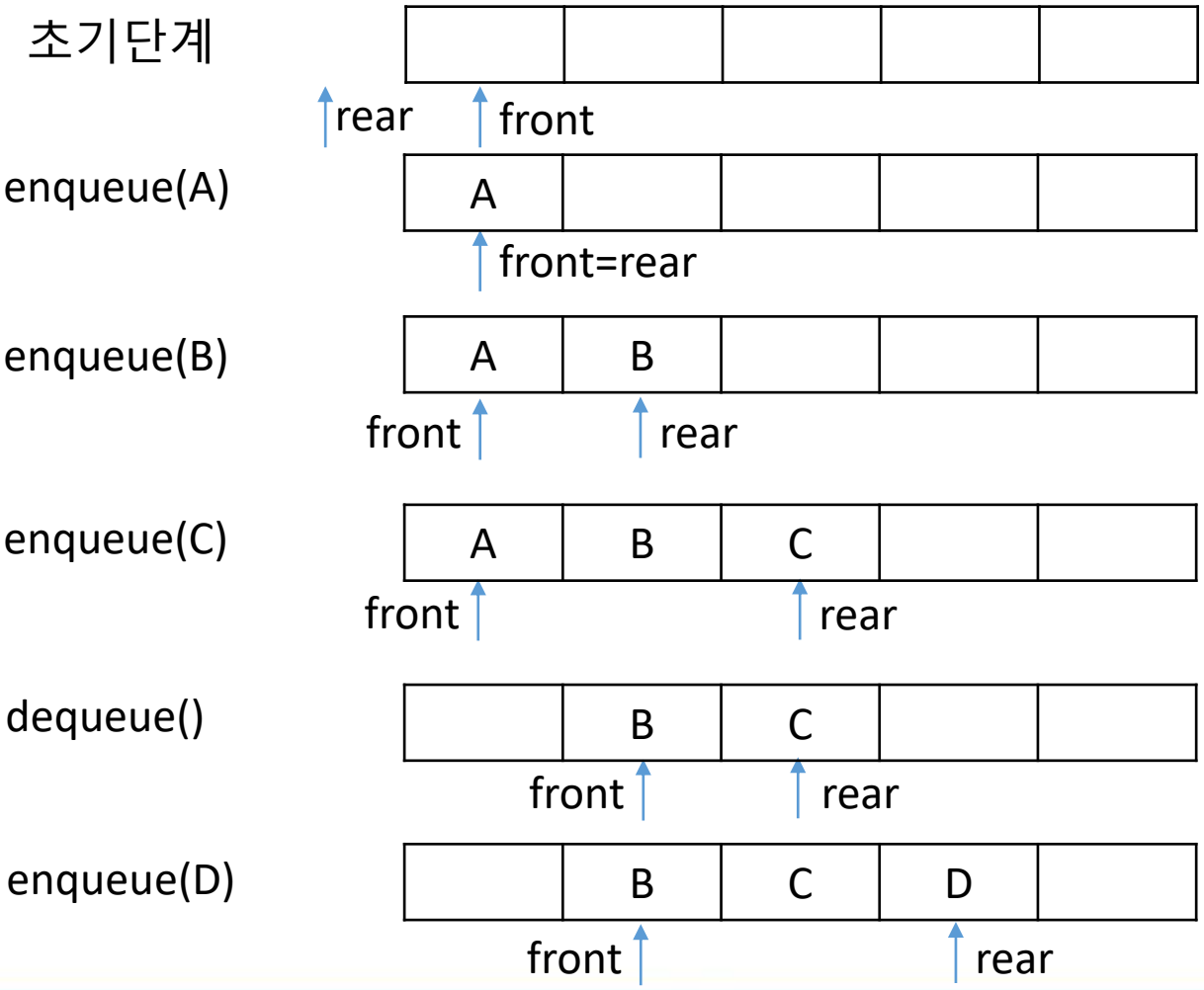


3 배열기반 선형 큐의 삽입과 삭제

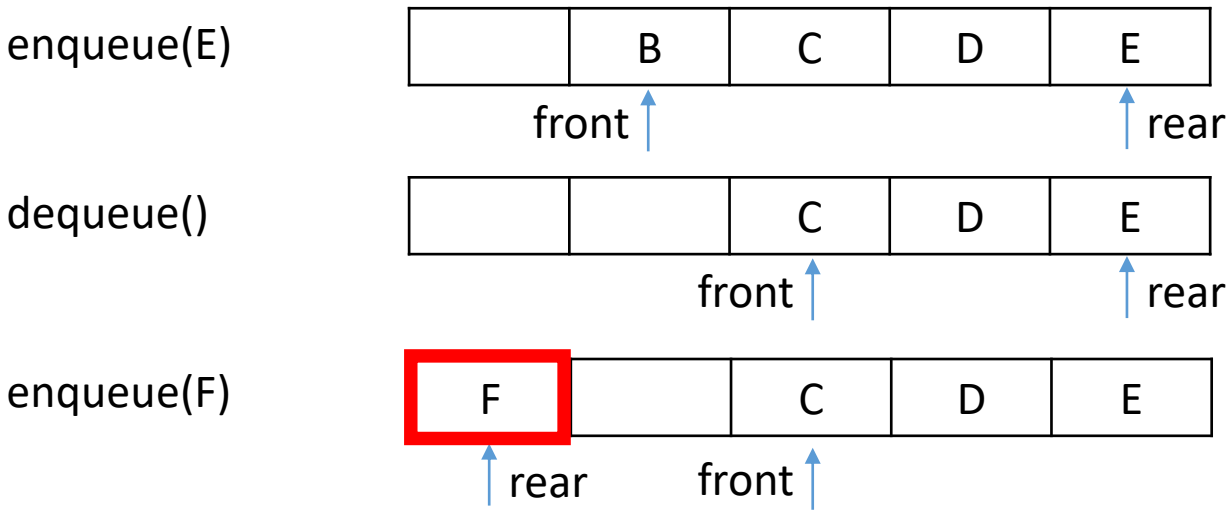


배열로 이루어진 선형 큐에서는 삽입과 삭제를 반복하면서 front, rear에 해당하는 위치가 배열의 크기를 넘어갈 수 없어, front 이전의 공간이 남아있어도 더 이상 사용할 수 없게 된다.

4 배열기반 원형 큐의 삽입과 삭제



4 배열기반 원형 큐의 삽입과 삭제



원형 큐는 남아있는 front 이전의 공간을 사용하여 선형 큐보다 메모리를 효율적으로 사용할 수 있는 구조이다.
단, 큐가 비어 있는 경우와 큐가 꽉 차 있는 두 경우 모두 front와 rear이 같은 곳을 지칭하게 된다. 이를 구분하기 위하여 배열의 1칸은 비워둔다.

4 배열기반 원형 큐의 삽입과 삭제

큐의 원소의 개수를 구하는 방법
rear >= front인 경우 : size = rear – front + 1
rear < front인 경우 : size = rear – front + 1 + capacity
종합 : size = (rear – front + 1 + capacity) mod (capacity)

큐(배열기반)

```
#include <iostream>
#include <string>
using namespace std;

class arrQueue {                        // 원형큐로 구현 (공간을 효과적으로 사용할 수 있음)
public:
    arrQueue(int size);                 // 생성자
    int size();                         // 큐 안의 원소의 개수
    bool isEmpty();                     // 큐가 비었는지 확인
    int front();                        // 가장 오래전에 입력된 큐의 원소 출력
    int rear();                         // 가장 최근에 입력된 큐의 원소 출력
    void enqueue(int data);             // 큐에 원소 입력
    void dequeue();                     // 오래된 원소 제거
    int* Q;                             // 동적으로 만들 큐
    int capacity;                       // 큐의 용량
    int f;                              // front의 위치
    int r;                              // rear의 위치
};
```

큐(배열기반)

```
arrQueue::arrQueue(int size) {
    this->Q = new int[size];          // r=f-1인 경우를 empty로 정의하기 위해, 1칸은 비워둠.
    this->capacity = size;
    this->f = 0;                      // f가 가리키는 곳부터 큐의 원소 존재
    this->r = -1;                     // r이 가리키는 곳까지 큐의 원소 존재
}
int arrQueue::size() {
    return
}
bool arrQueue::isEmpty() {
    if
    else
}
int arrQueue::front() {
    if
    else
}
int arrQueue::rear() {
    if
    else
}
void arrQueue::enqueue(int data) {
    if
    else
}
void arrQueue::dequeue() {
    if
    else
}
```

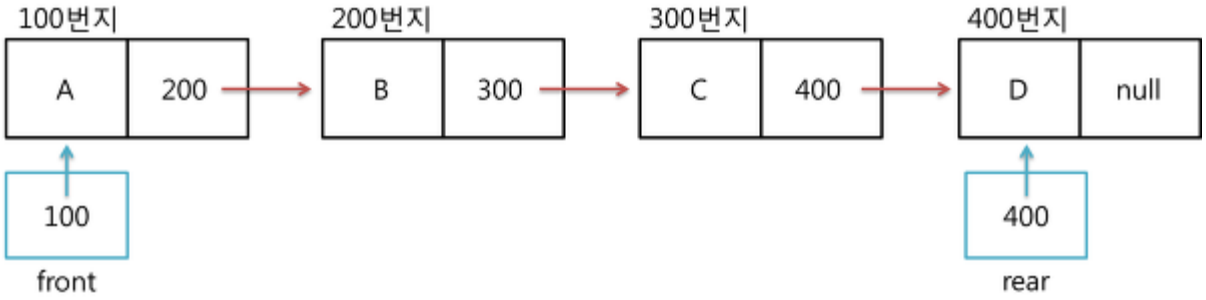
큐(배열기반)

```
int main( ) {
    int qSize;
    //cout << "Input your queue size : ";                    // 이해를 위한 주석코드
    cin >> qSize;
    arrQueue Q(qSize);
    int cmdNum;
    //cout << "How many commands do you want to insert : ";  // 이해를 위한 주석코드
    cin >> cmdNum;
    while (cmdNum--) {
        string cmd;
        cin >> cmd;
        if (cmd == "enqueue") {
            int input;
            cin >> input;


        }
        else if (cmd == "dequeue") {

        }
        else if (cmd == "size") {

        }
        else if (cmd == "isEmpty") {

        }
        else if (cmd == "front") {
            if
            else
        }
        else if (cmd == "rear") {
            if
            else
        }
        //else {
        //  cout << "Your command is wrong, try again." << endl;
        //}
    }
    //system("pause");                  // 제출시에는 주석처리, 코딩 중에 콘솔창이 바로 종료되는 경우에 잠시 주석처리 해제할것.
    return 0;
}
```

```
Input your queue size : 5
How many commands do you want to insert : 28
isEmpty
1
front
Empty
rear
Empty
size
0
enqueue 2
front
2
rear
2
size
1
dequeue
2
front
Empty
rear
Empty
size
0
dequeue
Empty
front
Empty
rear
Empty
size
0
enqueue 5
front
5
rear
5
size
1
isEmpty
0
front
5
rear
5
size
1
enqueue 3
front
5
rear
3
size
2
계속하려면 아무 키나 누르십시오 . . .
```

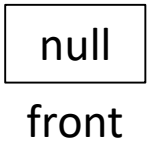
큐(연결리스트 기반)

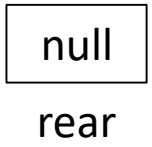
4 연결리스트 기반 큐



5 리스트 기반 큐의 삽입

초기 단계





앞과 뒤를 명시할 포인터 front, rear을 생성

큐(연결리스트 기반)

5 리스트 기반 큐의 삽입

초기 단계

enqueue 5

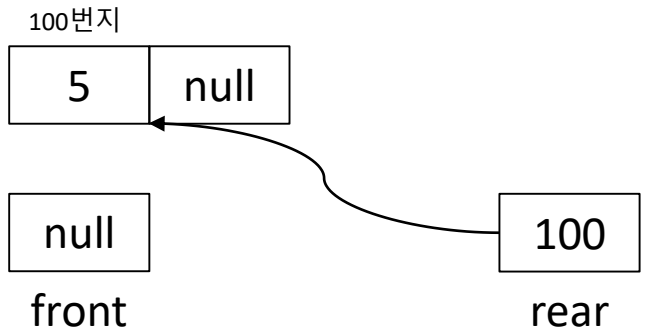
삽입할 데이터를 위해
새로운 노드 생성

큐(연결리스트 기반)

5 리스트 기반 큐의 삽입

초기 단계

enqueue 5



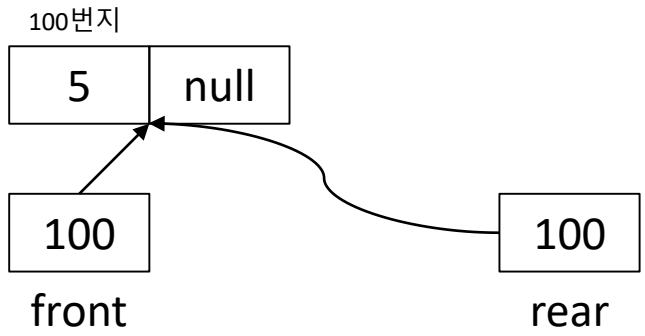
rear 포인터의 위치 변경

큐(연결리스트 기반)

5 리스트 기반 큐의 삽입

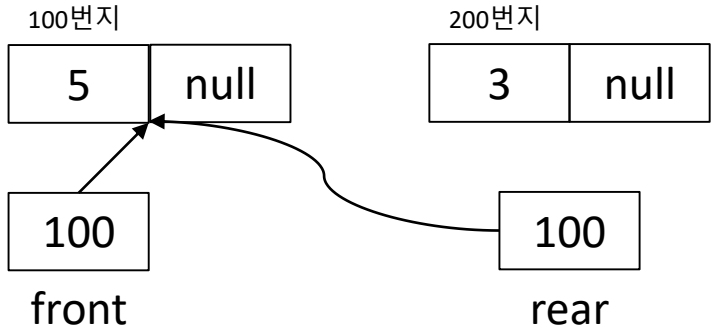
초기 단계

enqueue 5



큐가 비어있던 경우,
front 포인터도 위치 변경

enqueue 3



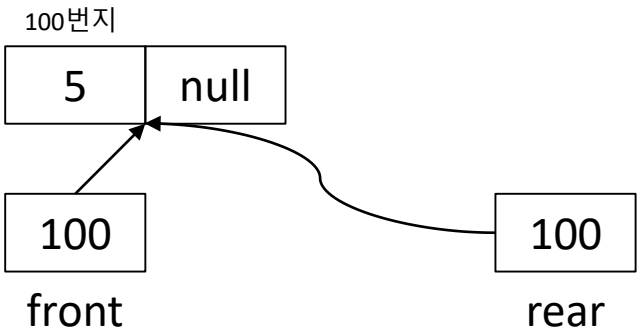
삽입할 데이터를 위해
새로운 노드 생성

큐(연결리스트 기반)

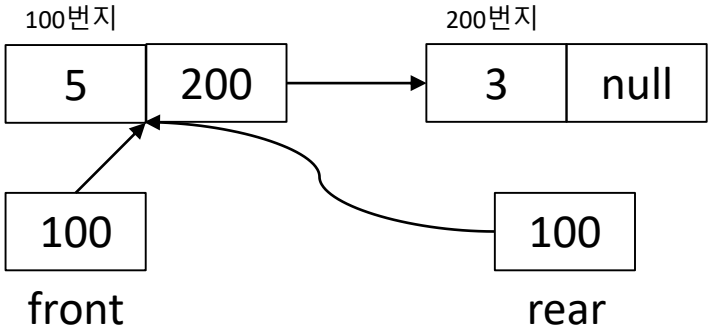
5 리스트 기반 큐의 삽입

초기 단계

enqueue 5



enqueue 3

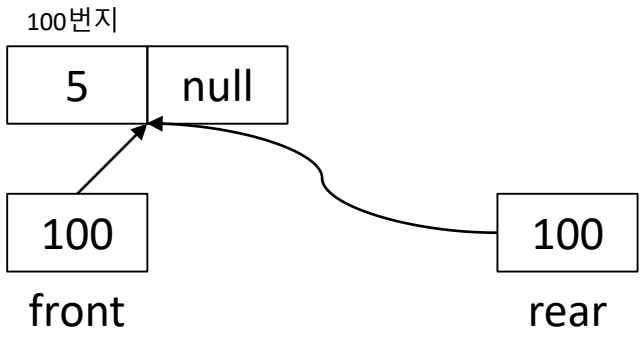


큐가 비어있지 않는 경우,
이전의 rear 포인터에 해당하는
노드의 next를 새로운 노드에 연결

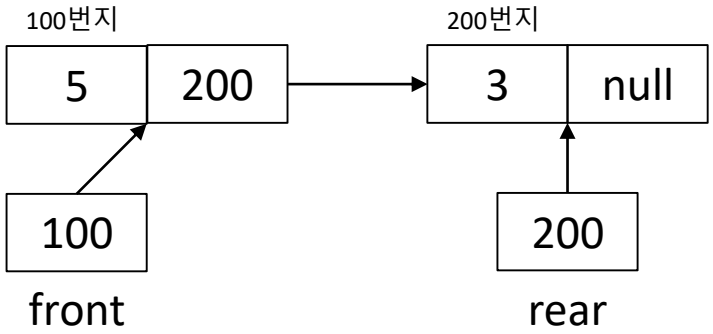
5 리스트 기반 큐의 삽입

초기 단계

enqueue 5

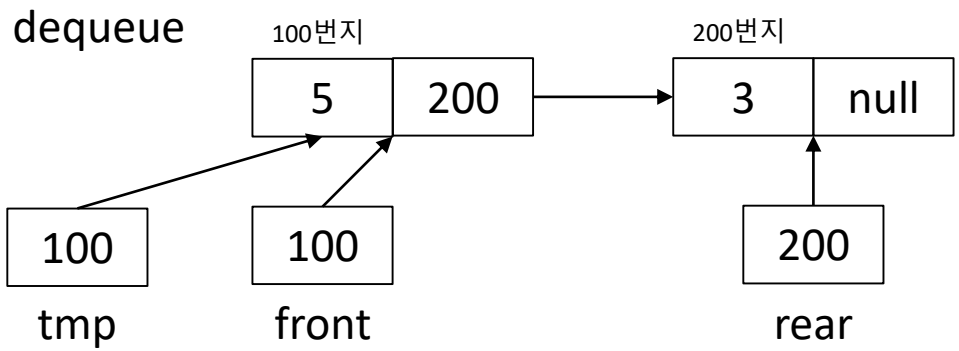


enqueue 3

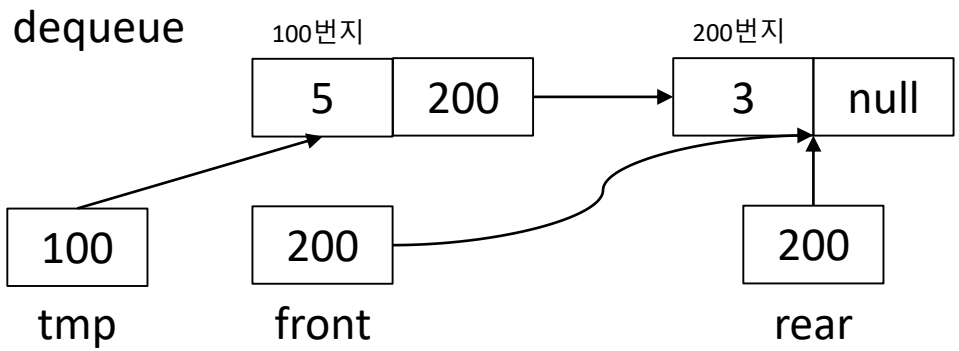


rear 포인터의 위치 변경

6 리스트 기반 큐의 삭제


dequeue
100번지
5
200
200번지
3
null
노드 제거를 위한,
임시 포인터 tmp 생성
및 front에 해당하는 노드 지칭
100
100
200
tmp
front
rear

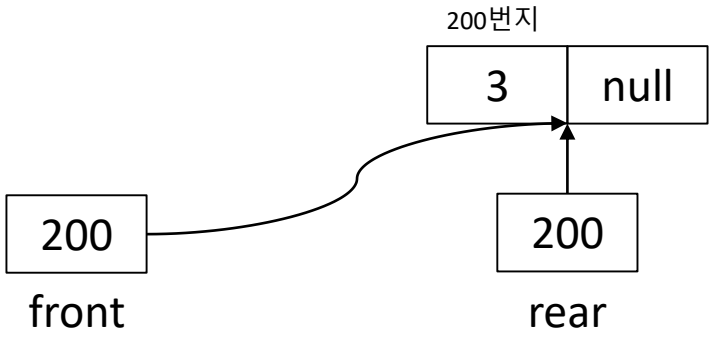
6 리스트 기반 큐의 삭제



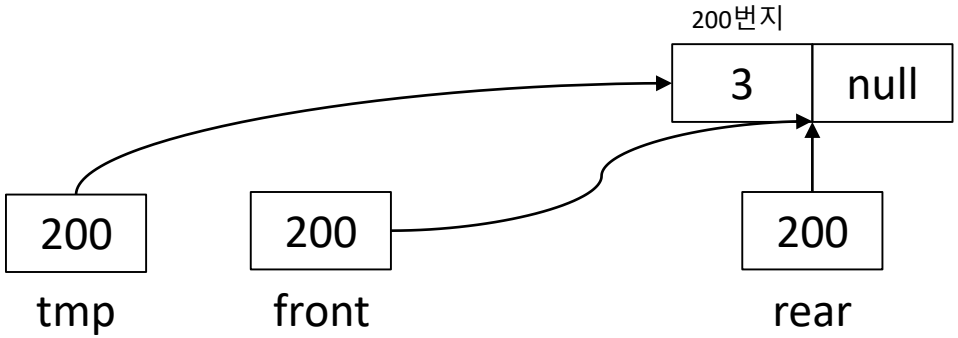
front포인터 위치변경
임시 포인터가 가리키는 노드와
임시 포인터 제거

6 리스트 기반 큐의 삭제

dequeue



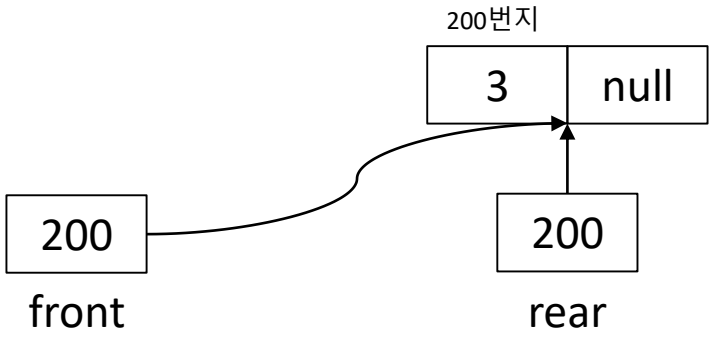
dequeue



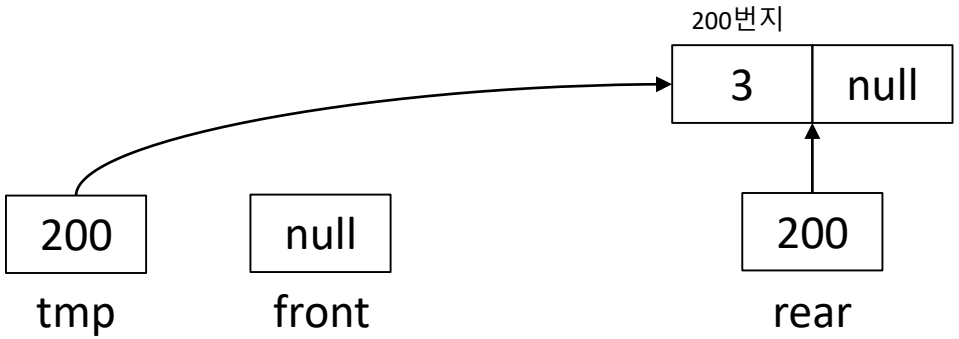
노드 제거를 위한,
임시 포인터 tmp 생성
및 front에 해당하는 노드 지칭

6 리스트 기반 큐의 삭제

dequeue



dequeue

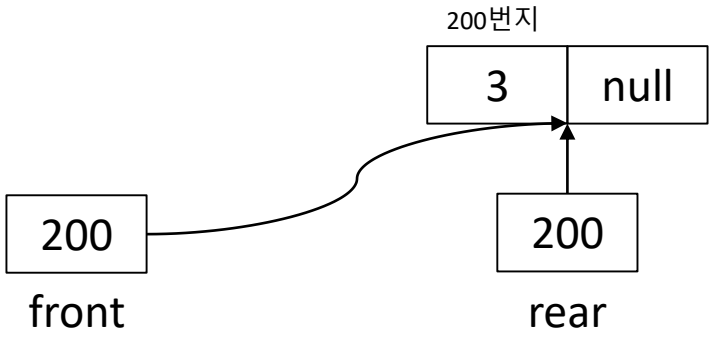


front포인터 위치변경
임시 포인터가 가리키는 노드와
임시 포인터 제거

큐(연결리스트 기반)

6 리스트 기반 큐의 삭제

dequeue

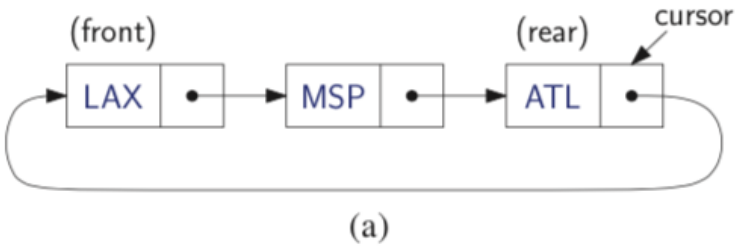


dequeue

큐가 비는 경우 rear도 null로 변환
(반드시 할 이유는 없음)

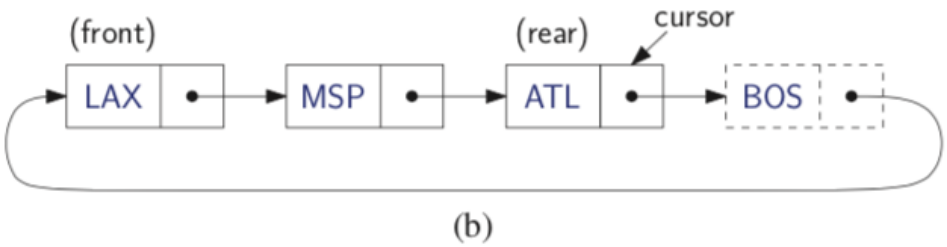
2 환형 링크드 리스트를 이용한 큐의 구현

1 환형 링크드 리스트로 구현된 큐에 'BOS' 삽입하기



(a)

(a) : 삽입하기 전



(b)

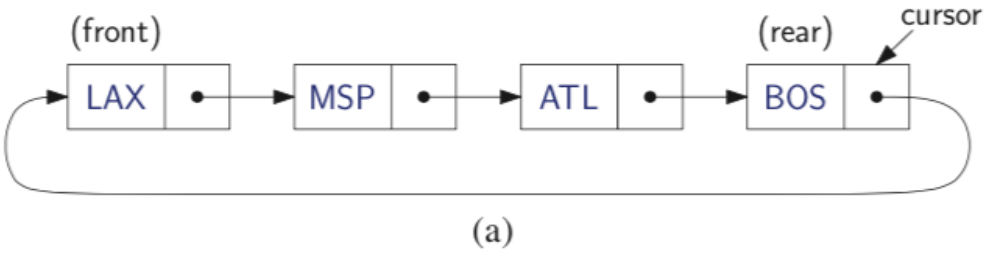
(b) : 새 노드를 삽입한 후

(c) : cursor를 뒤로 이동한 후

큐(연결리스트 기반)

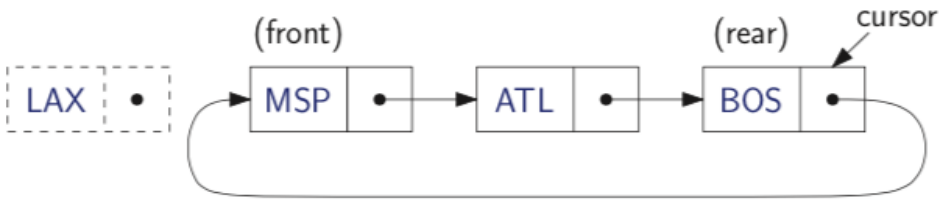
2 환형 링크드 리스트를 이용한 큐의 구현

2 환형 링크드 리스트로 구현된 큐의 앞에서 원소를 삭제하기 (LAX)



(a)

(a) : 삭제하기 전



(b)

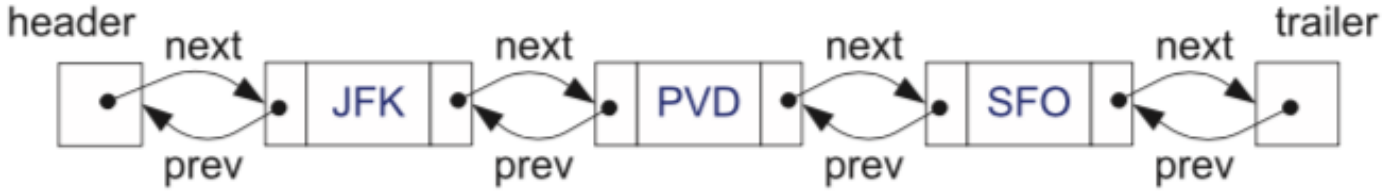
(b) : cursor 바로 다음 노드를 삭제한 후

3 양방향 큐 (데크)

1 양방향 큐

큐의 front와 rear 양쪽에서 삽입과 삭제연산을 지원하는 큐와 유사한 데이터 구조. 양방향 큐(double-ended queue) 또는 데크(deque)라고 부른다.

2 이중 링크드 리스트를 사용한 데크의 구현



리스트의 끝을 나타내는 센터널인 header와 trailer를 갖는 이중 연결 리스트. 데크의 첫 번째 원소(“JFK”)는 이 header의 바로 뒤에 저장되었고, 데크의 마지막 원소(“SFO”)는 이 trailer의 앞에 저장되었다.

큐(연결리스트 기반)

```
// 1주차에 배운, 링크드 리스트 내용
class Node {
public:
    Node(int data);             // 생성자는 멤버변수를 초기화할 때 사용한다.
    int data;
    Node* next;
};
Node::Node(int data) {
    this->data = data;          // this 함수를 사용하는 이유. 멤버변수와 매개변수를 구별한다.
                                // 매개변수의 data(우측)를 다른 이름으로 하면
                                // this를 사용할 이유는 없지만
    this->next = NULL;          // 의미를 확실하게 전달하기 위해 사용함.
}

class LinkedList {
public:
    Node* f;
    Node* r;
    LinkedList();
    int& front();
    int& end();
    void addEnd(int data);
    void removeFront();
};
class LinkedQueue {                     // LinkedList 기반으로 만드는 Queue.
public:
    LinkedList *S;                      // Queue를 동적으로 생성하기 위한 LinkedList형 포인터.
    int n;                              // 큐 안의 삽입된 원소의 개수
    LinkedQueue();                      // 생성자
    int size();                         // 현재 큐의 삽입된 원소의 개수를 확인하는 함수
    int isEmpty();                      // 큐가 비었는지 확인하는 함수
    int  front();                       // 큐의 맨 앞의 원소를 확인하는 함수
    int  rear();                        // 큐의 맨 뒤의 원소를 확인하는 함수
    void enqueue(int e);                // 큐의 맨 뒤에 원소 삽입
    void dequeue();                     // 큐의 맨 앞에 원소 제거
};
```

큐(연결리스트 기반)

```
LinkedList::LinkedList() {                  // 생성자를 이용하여 f,r을 초기화

}
int LinkedList::front() {                   // 맨 앞에 있는 Node의 data값을 반환
    return
}
int LinkedList::end() {                   // 맨 뒤에 있는 Node의 data값을 반환
    return
}
void LinkedList::addEnd(int data) {       // 맨 뒤에 data값을 갖는 Node를 추가
    Node* newNode = new Node(data);       // 새로운 Node를 추가하고
    if (f == NULL) {                          // 만약 List에 Node 추가가 처음이라면

    }
    else {                                    // 처음이 아닌 경우

    }
}
void LinkedList::removeFront() {          // 맨 앞의 Node를 제거

    //if (f == NULL) r = NULL;
}
```

큐(연결리스트 기반)

```
LinkedQueue::LinkedQueue( ) {              // 리스트 기반의 queue 생성 및 삽입된 개수 초기화하는 생성자.

}
int LinkedQueue::size( ) {                 // 삽입된 개수 반환하는 함수.

}
int LinkedQueue::isEmpty( ) {              // 큐의 원소가 비어있는지 확인하는 함수.
    if
    else
}
int LinkedQueue::front( ) {                // 큐의 맨 앞의 원소를 반환. 없는 경우 Empty를 출력.
    if
    else
}
int LinkedQueue::rear( ) {                 // 큐의 맨 뒤의 원소를 반환. 없는 경우 Empty를 출력.
    if
    else return
}
void LinkedQueue::enqueue(int e) {         // 큐의 맨 뒤에 원소를 추가.

}
void LinkedQueue::dequeue( ) {             // 큐의 맨 앞의 원소를 제거. 없는 경우 Empty를 출력.
    if
    else
}
```

큐(연결리스트 기반)

```
int main( ) {
    int N;
    cin >> N;
    LinkedQueue Q;
    while (N--) {
        string cmd;
        cin >> cmd;
        if (cmd == "enqueue") {
            int input;
            cin >> input;

        }
        else if (cmd == "dequeue") {

        }
        else if (cmd == "size") {

        }
        else if (cmd == "isEmpty") {

        }
        else if (cmd == "front") {
            if
            else
        }
        else if (cmd == "rear") {
            if
            else
        }
        //else {
        //  cout << "Your command is wrong, try again." << endl;
        //}
    }
    //system("pause");
    return 0;
}
```

```
28
isEmpty
1
front
Empty
rear
Empty
size
0
enqueue 2
front
2
rear
2
size
1
dequeue
2
front
Empty
rear
Empty
size
0
dequeue
Empty
front
Empty
rear
Empty
size
0
enqueue 5
front
5
rear
5
size
1
isEmpty
0
front
5
rear
5
size
1
enqueue 3
front
5
rear
3
size
2
계속하려면 아무 키나 누르십시오 . . .
```


Thank you